Fuzzy strings matching – rozmyte dopasowanie wzorca

Definicja

Rozmyte dopasowanie wzorca to technika polegająca na przybliżonym porównaniu dwóch tekstów. Pozwala na określenie podobieństwa pomiędzy dwoma tekstami. Do określenia podobieństwa wykorzystywane mogą być różne funkcje, od których zależy obliczone podobieństwo.

“jabłko” ≈ “jablko”
“microsoft” ≈ “micro$oft”

Funkcje do określania podobieństwa tekstu

Odległość Hamminga
Odległość Levenshteina
Odległość Jaro-Winkler
Indeks Jaccarda

Prosta funkcja opracowana przez Richarda Hamminga, pozwala na porównanie ciągów znaków o tej samej długości. Algorytm uwzględnia tylko równość znaków na tych samych pozycjach.

Porównanie	zagrabić zagrabić	zagrabić zatrąbił	zagrabić zagrabione	Tom Marvolo Riddle I am Lord Voldemort
Odległość	0	3	error	14

Funkcje do określania podobieństwa tekstu

Odległość Hamminga
Odległość Levenshteina
Odległość Jaro-Winkler
Indeks Jaccarda

Odległość Levenshteina jest rozwinięciem odległości Hamminga. Dzięki uwzględnieniu możliwości dodawania i usuwania znaków, pozwala na porównanie tekstów o różnej długości.

Porównanie	zagrabić zagrabić	zagrabić zatrąbił	zagrabić zagrabione	Tom Marvolo Riddle I am Lord Voldemort
Odległość	0	3	3	12

Funkcje do określania podobieństwa tekstu

Odległość Hamminga
Odległość Levenshteina
Odległość Jaro-Winkler
Indeks Jaccarda

Odległość Jaro to miernik podobieństwa między dwoma ciągami znaków, który został opracowany przez amerykańskiego informatyka, Matthew A. Jaro'a. Różni się od odległości Levensteina tym, że pozwala na transpozycję znaków znajdujących się odpowiednio blisko siebie. William E. Winkler dodał do algorytmu uwzględnianie prefixów ciągów, które mogą dodatkowo zwiększyć wynik. Sam wynik jest normalizowany, stąd wartości od 0 do 1.

Porównanie	`zagrabić` `zagrabić`	`zagrabić` `zatrąbił`	`zagrabić` `zagrabione`	`Tom Marvolo Riddle` `I am Lord Voldemort`
Odległość	1.0	0.916667	0.708333	0.67773

Funkcje do określania podobieństwa tekstu

Odległość Hamminga
Odległość Levenshteina
Odległość Jaro-Winkler
Indeks Jaccarda

Indeks Jaccarda to miernik podobieństwa między dwoma zbiorami, który został opracowany przez francuskiego matematyka i statystyka, Paul'a Jaccard'a. Jego wartość jest obliczana jako stosunek liczby elementów wspólnych obu zbiorów do liczby elementów, które są w obu zbiorach. Wartość indeksu Jaccarda może wahać się od 0 do 1, gdzie wartość 0 oznacza brak elementów wspólnych między zbiorami, a wartość 1 oznacza, że zbiory są identyczne.

Porównanie	zagrabić zagrabić	zagrabić zatrąbił	zagrabić zagrabione	Tom Marvolo Riddle I am Lord Voldemort
Odległość	1	0.5	0.6	1

Pakiet TheFuzz

TheFuzz to pakiet Pythona, który wykorzystuje połączenie logiki rozmytej i odległości Levenshteina, aby znaleźć podobieństwo między dwoma ciągami. Można go użyć do porównania dwóch ciągów i określenia, jak bardzo są one podobne, nawet jeśli nie mają tej samej pisowni lub są nieco inne. Dzięki temu jest przydatny do zadań takich jak sprawdzanie pisowni, dopasowywanie podobnych nazwisk i nie tylko.

```
from thefuzz import fuzz


string1 = "this is a test"
string2 = "this is a test!"


print(f'Ratio {fuzz.ratio(string1, string2)}')
print(f'Partial ratio {fuzz.partial_ratio(string1, string2)}')
```

```
C:\Users\przem\Desktop>python fuzzywuzzy.py
Ratio 97
Partial ratio 100
```

Przykład realizacji algorytmu rozmytego dopasowania wzorca na przykładzie odległości Levenshteina

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \operatorname{lev}\left(\operatorname{tail}(a), b\right) \\ \operatorname{lev}\left(a, \operatorname{tail}(b)\right) \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) \end{cases} & \text{otherwise,} \end{cases}$$

Źródło: https://en.wikipedia.org/wiki/Levenshtein_distance

Przykład realizacji algorytmu rozmytego dopasowania wzorca na przykładzie odległości Levenshteina

```python
def levenstein_recursive(word1: str, word2: str):
    # if end of one of strings, then return remaining difference
    if len(word1) == 0 or len(word2) == 0:
        return max(len(word1), len(word2))
    # first  characters match, check rest
    if word1[0] == word2[0]:
        return levenstein_recursive(word1[1:], word2[1:])

    # fist characters don't match, check possible actions
    return 1 + min(levenstein_recursive(word1[1:], word2),
                   levenstein_recursive(word1, word2[1:]),
                   levenstein_recursive(word1[1:], word2[1:]))
```

Rekursywne obliczanie odległości Levensteina, bardzo mało wydajne

Przykład realizacji algorytmu rozmytego dopasowania wzorca na przykładzie odległości Levenshteina

```python
def levenstein_matrix(word1: str, word2: str):
    word1_len: int = len(word1)
    word2_len: int = len(word2)
    matrix: np.ndarray = np.zeros((word1_len + 1, word2_len + 1))

    # fill matrix places for one string comparison with empty string
    for i in range(word1_len + 1):
        matrix[i][0] = i
    for i in range(word2_len + 1):
        matrix[0][i] = i

    # fill matrix, by finding best action and adding its result to
    # previous best action result
    for i in range(1, word1_len + 1):
        for j in range(1, word2_len + 1):
            if word1[i - 1] == word2[j - 1]:
                substitution = 0
            else:
                substitution = 1

            matrix[i][j] = min(matrix[i - 1][j] + 1, matrix[i][j - 1] + 1,
                               matrix[i - 1][j - 1] + substitution)
    return matrix[word1_len][word2_len]
```

Obliczanie odległości Levensteina operując na macierzy , bardzo mało wydajne

Przykład realizacji algorytmu rozmytego dopasowania wzorca na przykładzie odległości Levenshteina

		i	a	m	l	o	r	d	v	o	l	d	e	m	o	r	t
[[	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.]
t	[1.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	11.	12.	13.	14.	15.]
o	[2.	2.	2.	3.	4.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.]
m	[3.	3.	3.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.]
m	[4.	4.	4.	3.	3.	4.	5.	6.	7.	7.	8.	9.	10.	11.	12.	13.	14.]
a	[5.	5.	4.	4.	4.	4.	5.	6.	6.	7.	7.	8.	9.	10.	11.	12.	13.]
r	[6.	6.	5.	5.	5.	5.	4.	5.	6.	7.	8.	7.	8.	9.	10.	11.	12.]
v	[7.	7.	6.	6.	6.	6.	5.	5.	6.	7.	8.	8.	8.	8.	9.	10.	11.]
o	[8.	8.	7.	7.	7.	7.	6.	5.	6.	7.	8.	9.	9.	9.	9.	10.	11.]
l	[9.	9.	8.	8.	8.	8.	7.	6.	5.	6.	7.	8.	9.	10.	10.	10.	11.]
o	[10.	10.	9.	9.	9.	9.	8.	7.	6.	5.	6.	7.	8.	9.	10.	10.	11.]
r	[11.	11.	10.	10.	10.	10.	9.	8.	7.	6.	6.	7.	8.	8.	9.	10.	11.]
i	[12.	12.	11.	11.	11.	11.	10.	9.	8.	7.	7.	7.	8.	9.	9.	10.	10.]
d	[13.	13.	12.	11.	11.	12.	11.	10.	9.	8.	8.	8.	8.	9.	10.	10.	11.]
d	[14.	14.	13.	12.	12.	12.	12.	11.	10.	9.	8.	9.	9.	9.	10.	11.	11.]
l	[15.	15.	14.	13.	13.	13.	12.	12.	11.	10.	9.	8.	9.	10.	10.	11.	12.]
e	[16.	15.	15.	14.	14.	14.	13.	13.	12.	11.	10.	9.	9.	10.	11.	11.	12.]]

Przykład macierzy powstającej przy obliczaniu odległości dla tekstów “iamlordvoldemort” i “tommarvoloriddle”

Bibliografia

https://en.wikipedia.org/wiki/Levenshtein_distance

https://www.geeksforgeeks.org/jaro-and-jaro-winkler-similarity/

https://en.wikipedia.org/wiki/Hamming_distance

https://en.wikipedia.org/wiki/Jaccard_index

https://github.com/seatgeek/thefuzz

Dziękujemy za uwagę!

Maciej Lebiest

Przemysław Marek